AVL Tree

AVL Tree

- AVL trees are height-balanced binary search trees
- Balance factor of a node=

  height(left sub tree) - height(right subtree)
- An AVL tree has balance factor calculated at every node
  - For every node, heights of left and right sub tree can differ by no more than 1
  - Store current heights in each node

AVL Tree

- If the balance factor of a node is 1, then it means that the left sub-tree of the tree is one level higher than that of the right sub-tree. Such a tree is therefore called as a ***left-heavy tree***.
- If the balance factor of a node is 0, then it means that the height of the left sub-tree (longest path in the left sub-tree) is equal to the height of the right sub-tree.
- If the balance factor of a node is –1, then it means that the left sub-tree of the tree is one level lower than that of the right sub-tree. Such a tree is therefore called as a ***right-heavy tree***.

Convert binary into AVL Tree

Step 1: Find the balanced factor(BF) of existing tree

BF for N = height of left sub tree – height of right sub tree

Step 2: Insert new node as per binary search tree rule.

Step 3: Check BF again after insertion

If BF is not [-1,0,1] then imbalanced is there.

Step 4: From newly inserted node to the root traverse back, the first node encountered which is not balanced is called critical node(CN).

Convert binary into AVL Tree

<u>Step 5:</u> Now critical node is found, find the which rotation is required to balanced the tree.(Only consider tree rooted with critical node.)

Step 6: If newly inserted node is in

Left sub-tree of Left sub-tree ------**LL**
Right sub-tree of Right sub-tree---**RR**

Go One Step in Same Direction.

Left sub-tree of Right sub-tree-----**LR**
Right sub-tree of Left sub-tree ----**RL**

Go two Step in Same Direction.

Rotation

- ***LL rotation*** The new node is inserted in the left sub-tree of the left sub-tree of the critical node.
- ***RR rotation*** The new node is inserted in the right sub-tree of the right sub-tree of the critical node.
- ***LR rotation*** The new node is inserted in the right sub-tree of the left sub-tree of the critical node.
- ***RL rotation*** The new node is inserted in the left sub-tree of the right sub-tree of the critical node.

Convert binary into AVL Tree

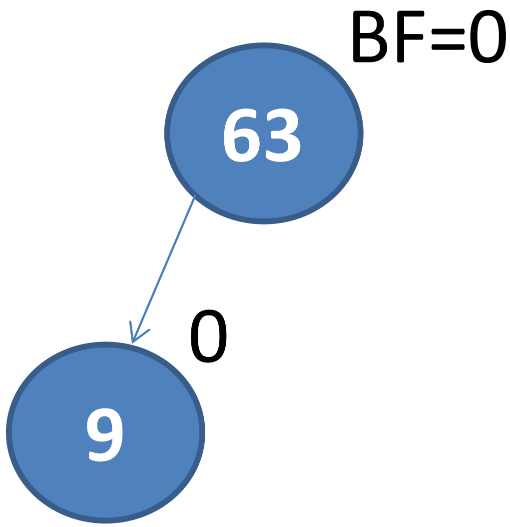
Step 7: From CN traverse as shown (Rule no. 6) the node where you stop is the new root of the tree rooted with CN.

Step 8: Find immediate Childs of new root for that (new root node) from node where you stop in step 7 traverse back to critical node, the node you visit on the path will be immediate children of new root.

Step 9: Arrange remaining node as per BST.

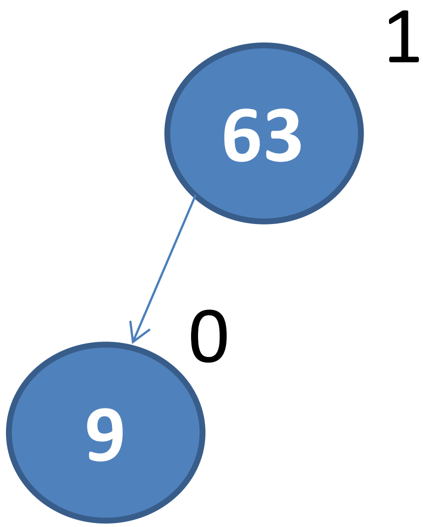
Example

63,9,19,70,50,55,53,52,

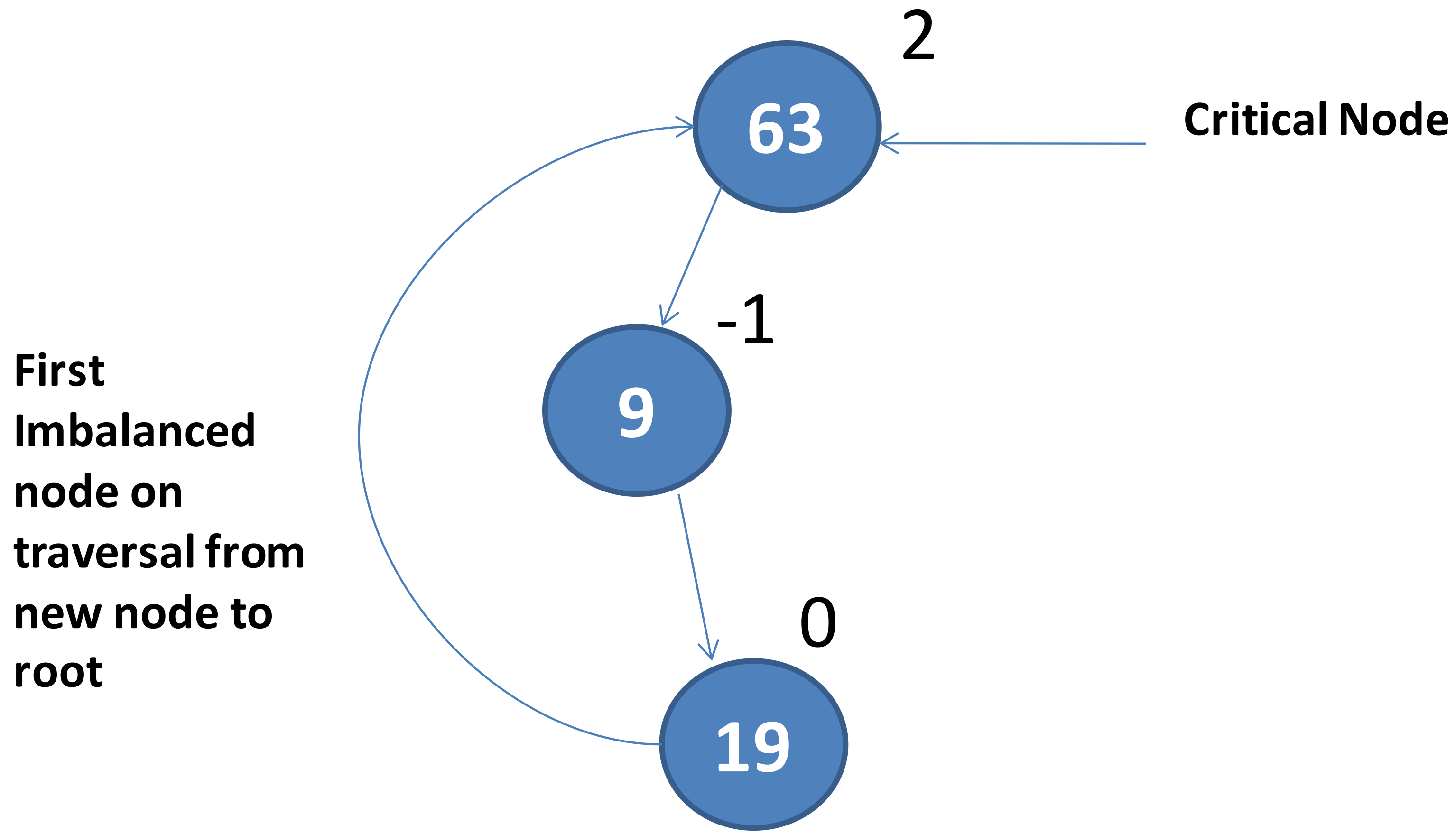


Example

63,9,19,70,50,55,53,52,

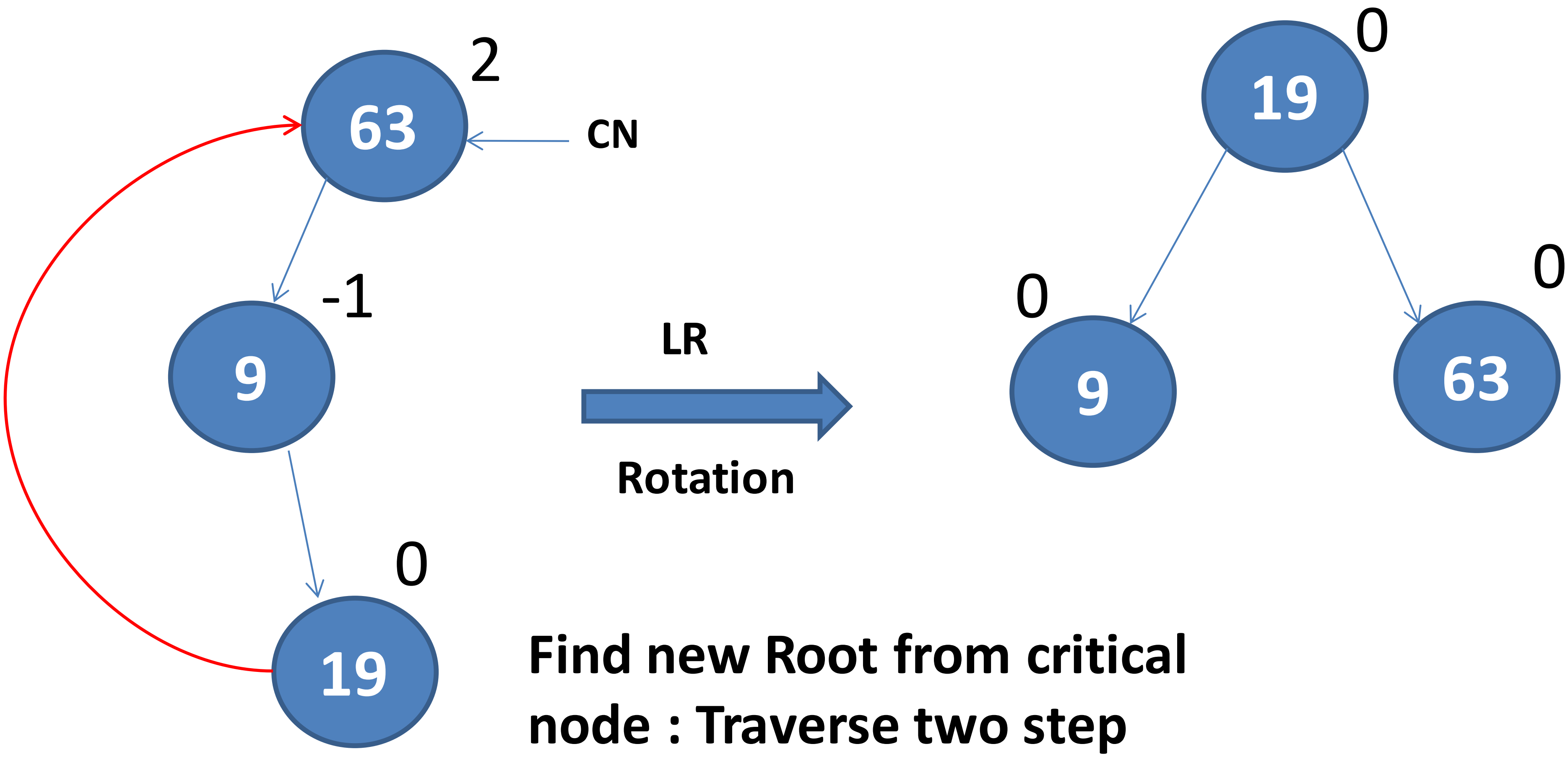


Example
63,9,19,70,50,55,53,52,



Example

63,9,19,70,50,55,53,52,

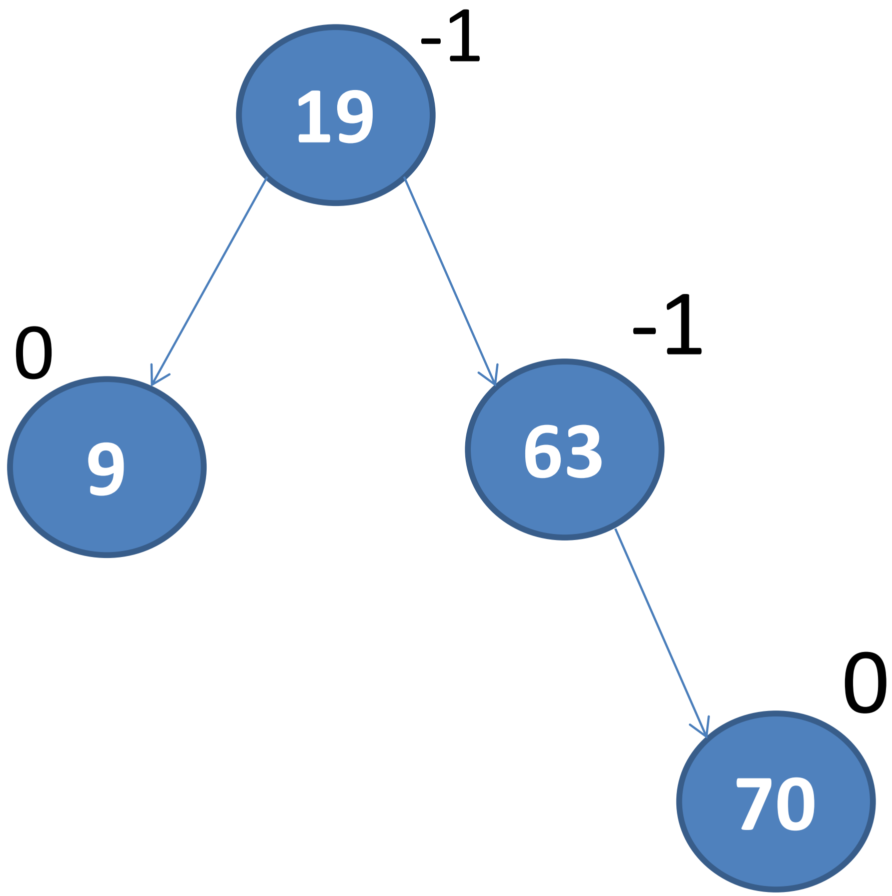


Find new Root from critical node : Traverse two step down.
So (19) is new root for subtree rooted with critical node (63)

Example

63,9,19,70,50,55,53,52,

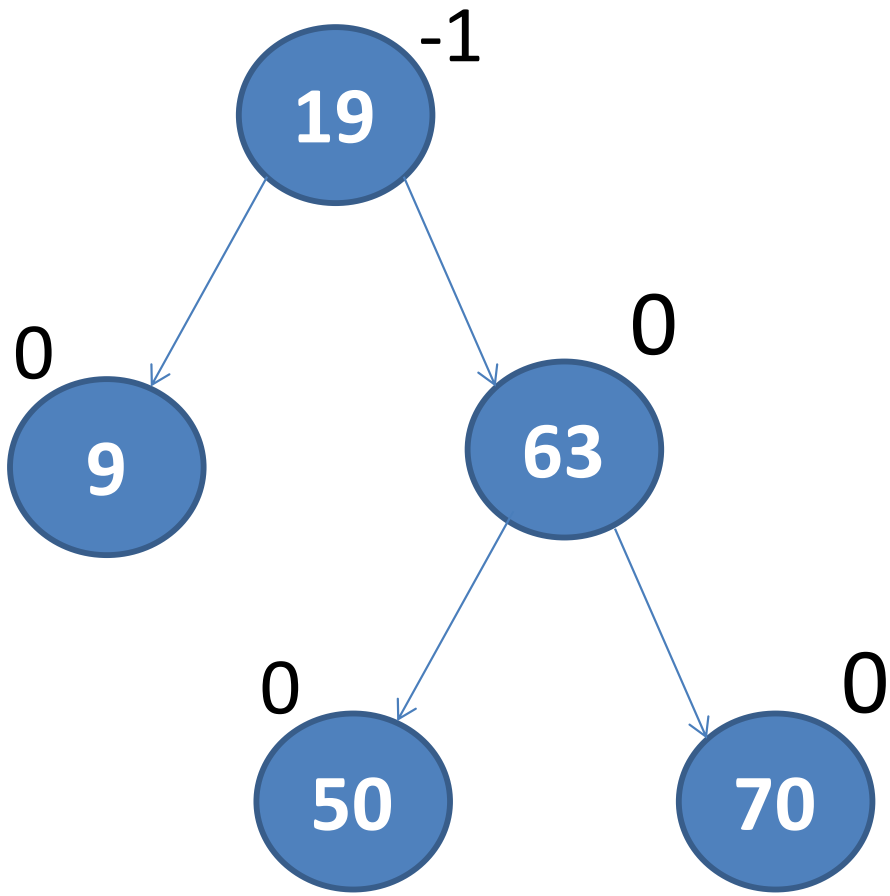
Next Input = 70



Example

63,9,19,70,50,55,53,52,

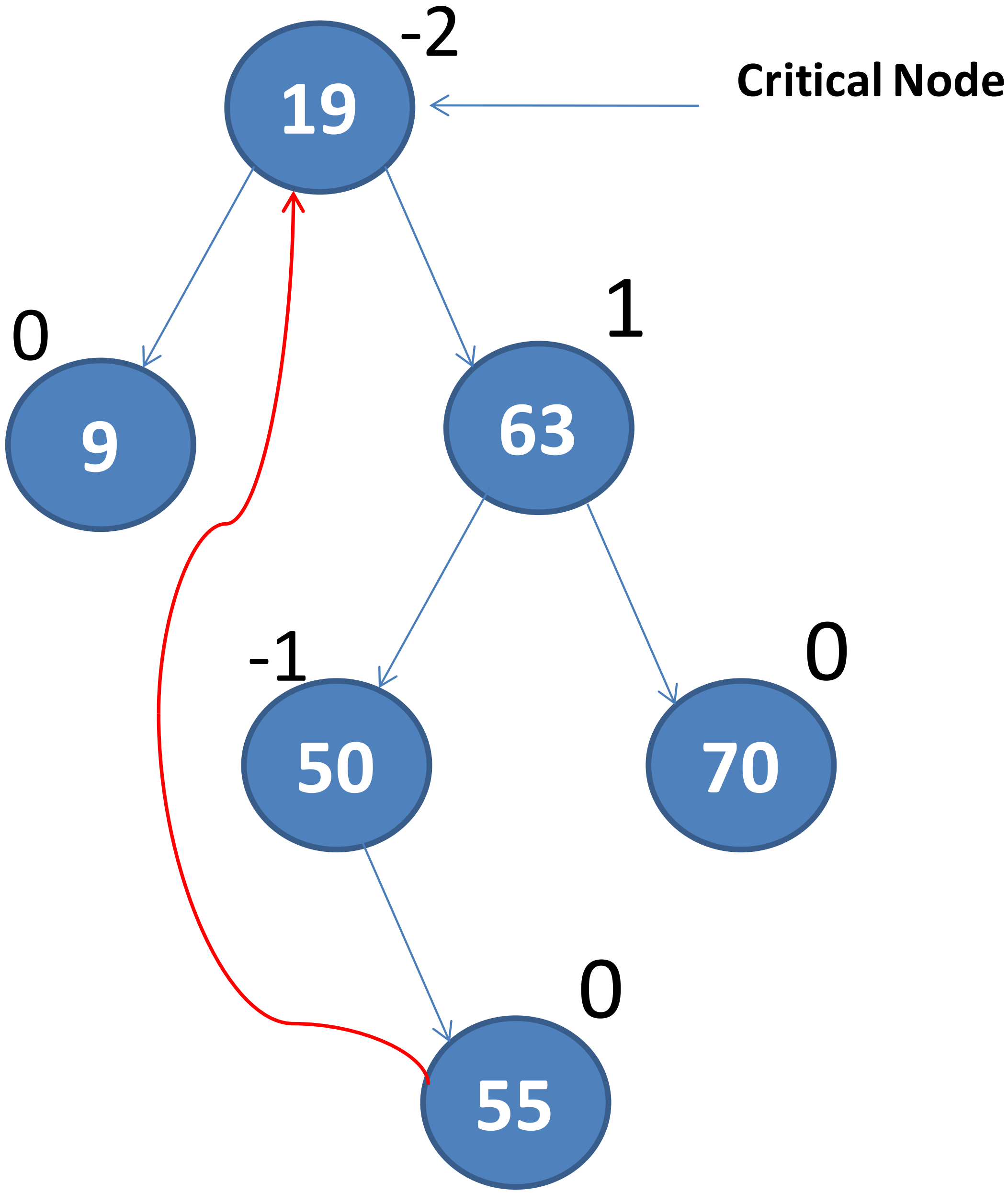
Next Input = 50



Example

63,9,19,70,50,55,53,52,

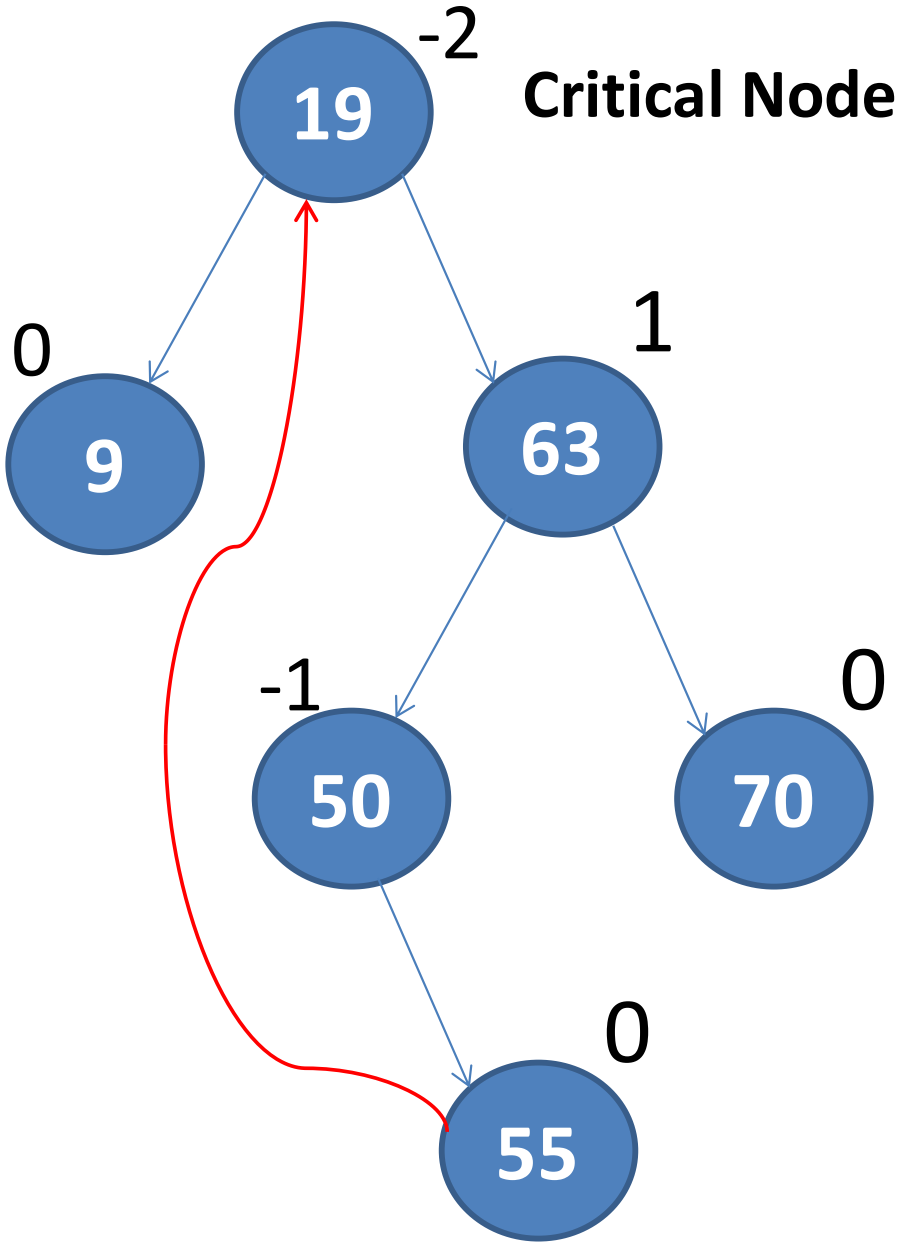
Next Input = 55



Example

63,9,19,70,50,55,53,52,

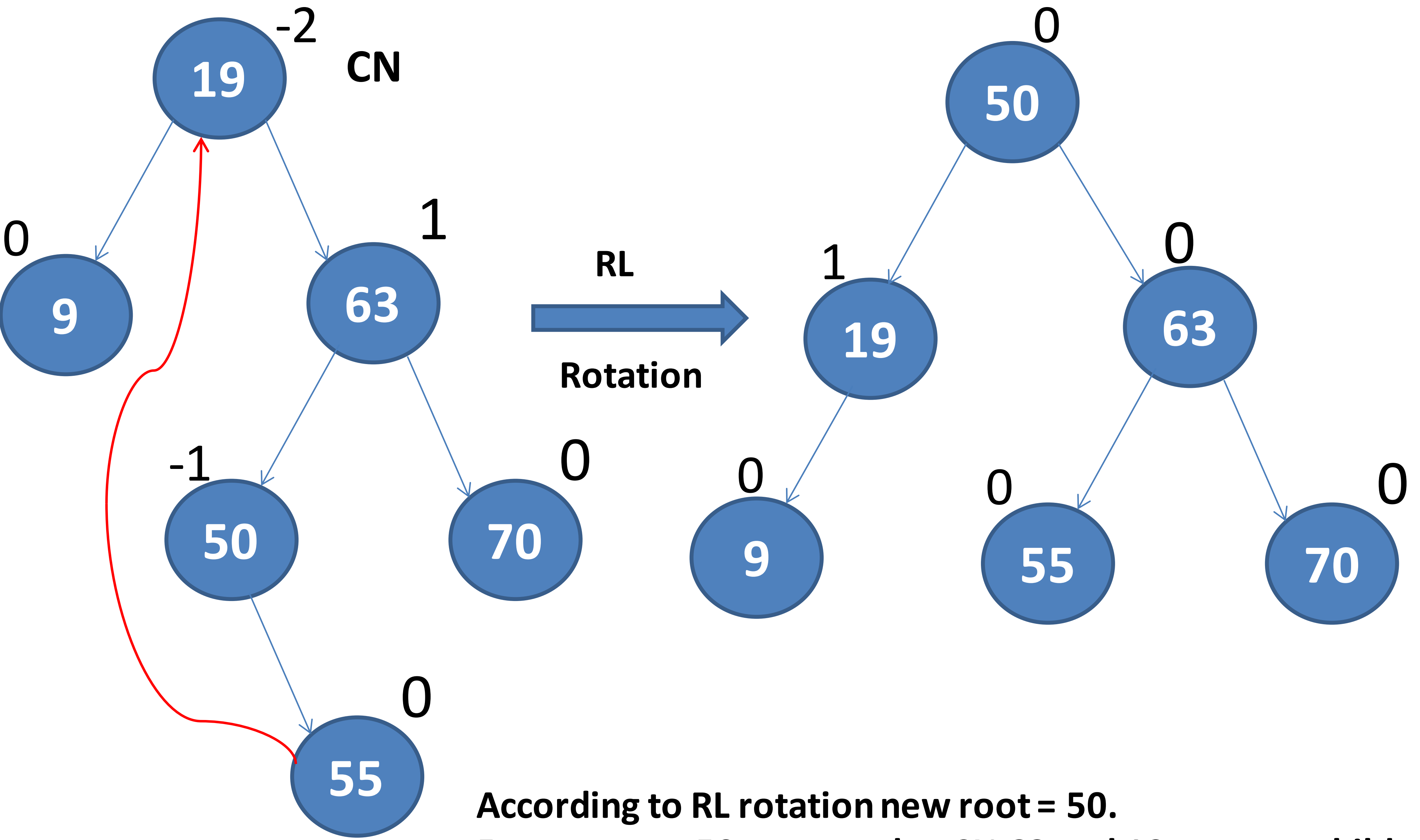
Next Input = 55



Go two step down from critical node, so it is RL rotation.

Example

63,9,19,70,50,55,53,52,

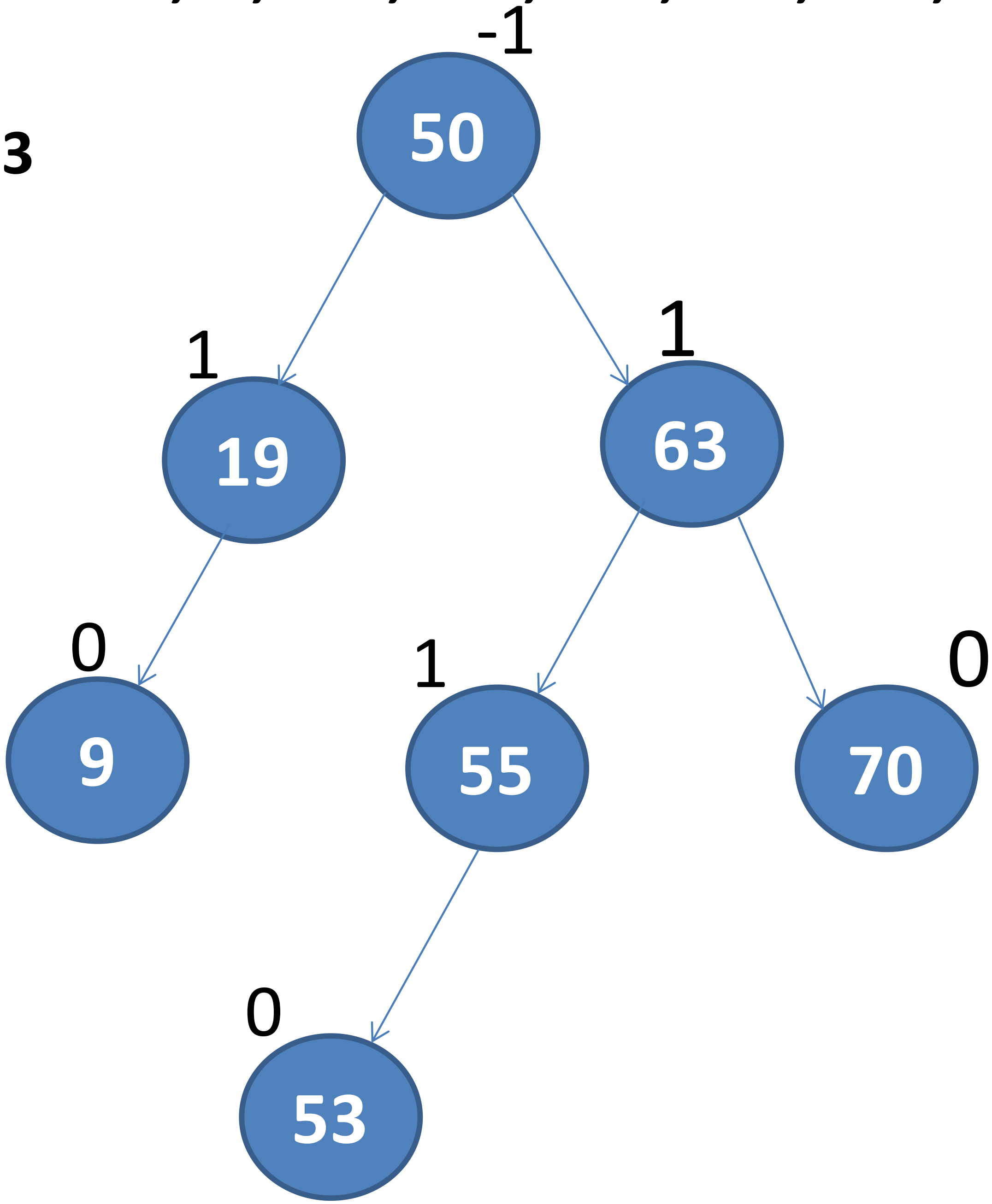


According to RL rotation new root = 50.
For new root 50 traversed to CN 63 and 19 are new child

Example

63,9,19,70,50,55,53,52,

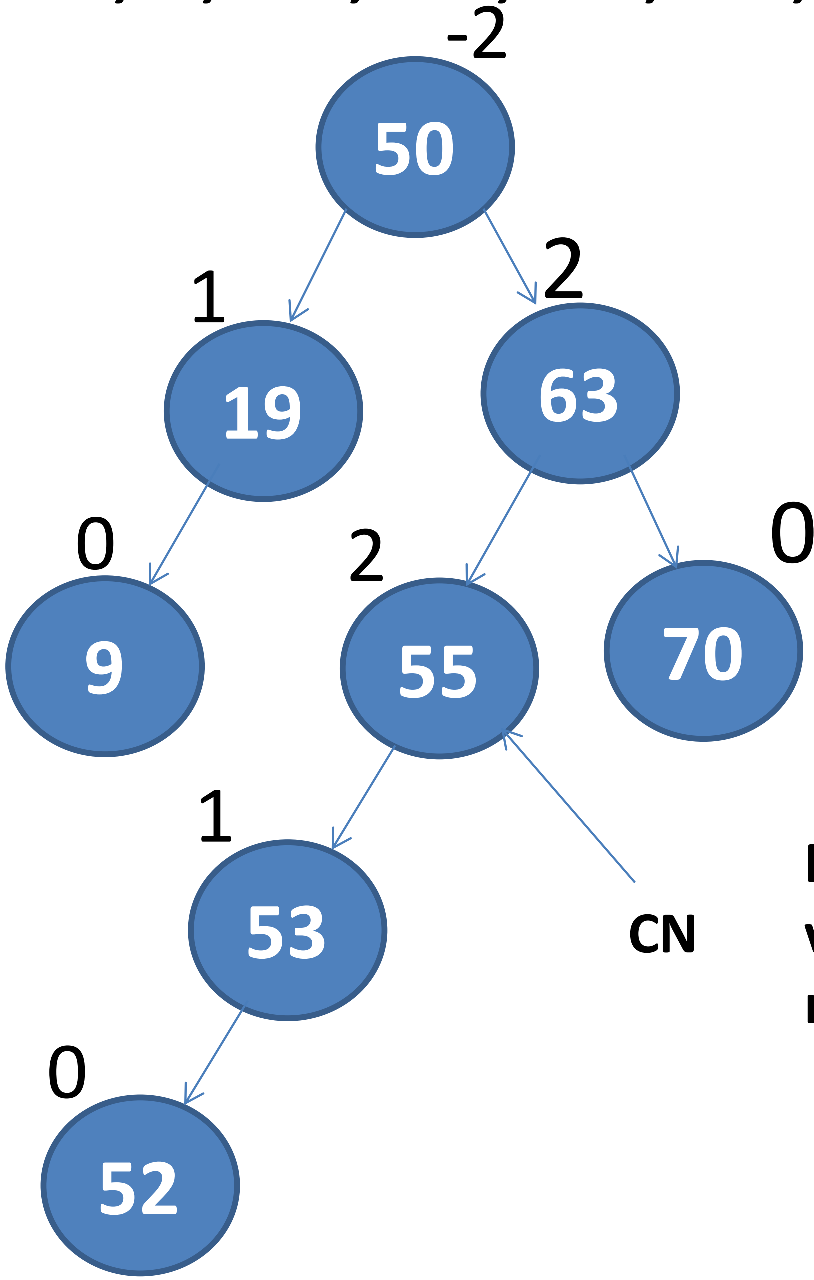
Next Input = 53



Example

63,9,19,70,50,55,53,52,

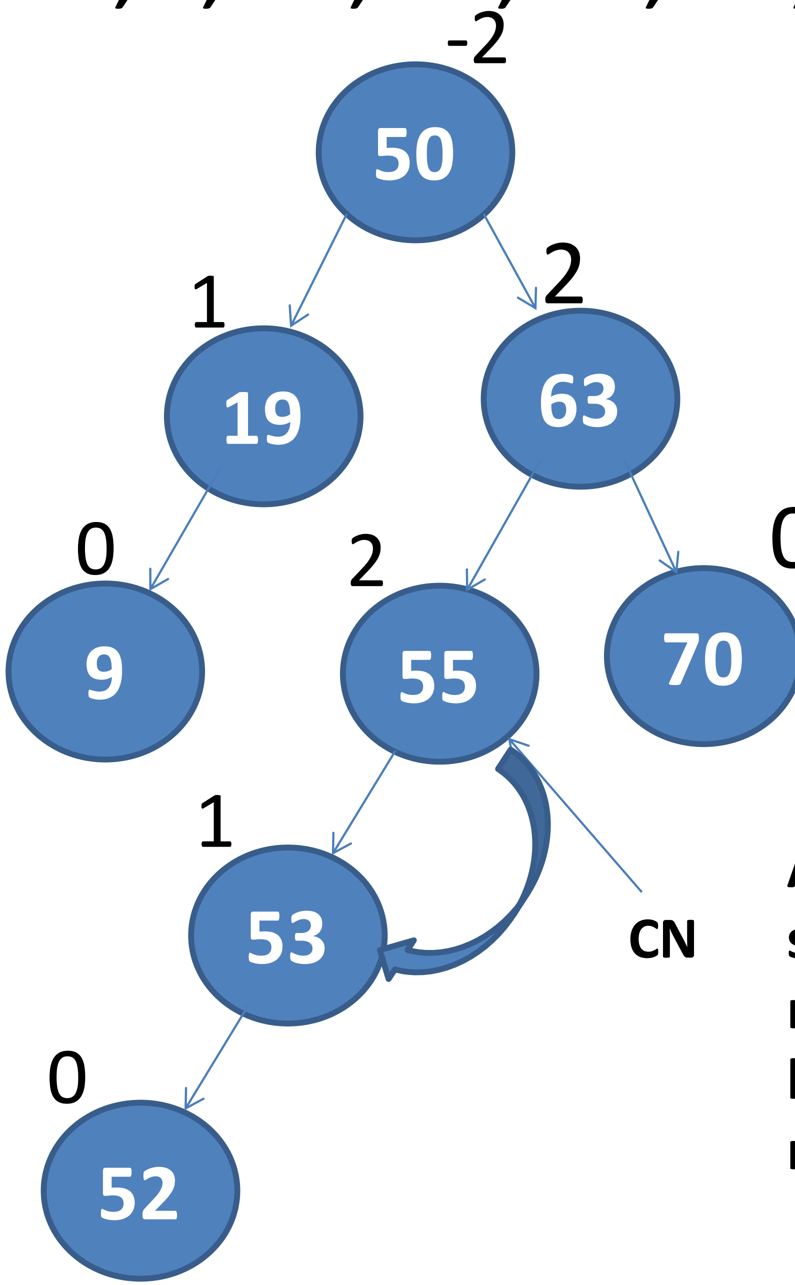
Next Input = 52



From the critical node it is visible that LL rotation is needed.

Example

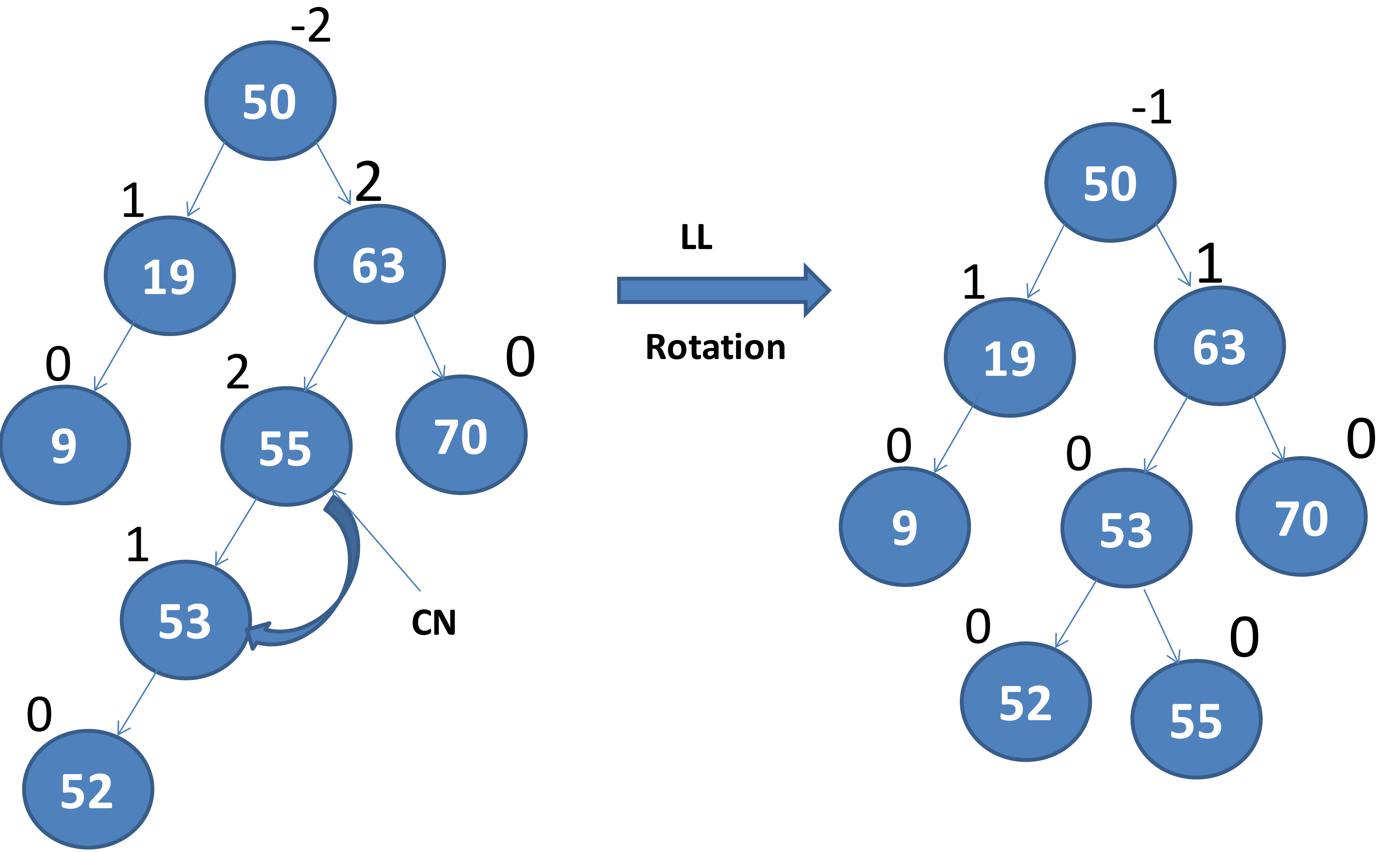
63,9,19,70,50,55,53,52,



According to LL rotation go one step down. So new root node=53 then 52 will become left child and 55 will become right child.

Example

63,9,19,70,50,55,53,52,



- The key advantage of using an AVL tree is that it takes O(log n) time to perform search, insert, and delete operations in an average case as well as the worst case because the height of the tree is limited to O(log n).

Deleting node from AVL tree

- Deletion of a node in an AVL tree is similar to that of binary search trees.
- But it goes one step ahead.
- Deletion may disturb the AVLness of the tree, so to rebalance the AVL tree, we need to perform rotations.
- There are two classes of rotations that can be performed on an AVL tree after deleting a given node.
  - R rotation
  - L rotation.

Rotations for Deletion

- On deletion of node X from the AVL tree, if node A becomes the critical node, then the type of rotation depends on whether X is in the left sub-tree of A or in its right sub-tree.
- node to be deleted is present in the left sub-tree of A, then L rotation is applied, else if X is in the right sub-tree, R rotation is performed.
- Further, there are three categories of L and R rotations.
  - The variations of L rotation are L–1, L0, and L1 rotation.
  - Correspondingly for R rotation, there are R0, R–1, and R1 rotations.